A compact data structure and parallel algorithms for permutation graphs

Jens Gustedt, Michel Morvan, Laurent Viennot

A Compact Data Structure and Parallel Algorithms for Permutation Graphs*

Jens Gustedt[1]**, Michel Morvan[2]***, and Laurent Viennot[2]†

[1] Technische Universität Berlin,
Sekr. MA 6-1, D-10623 Berlin - Germany.
[2] LITP/IBP Université Paris 7 Denis Diderot,
Case 7014, 2, place Jussieu F-75251, Paris Cedex 05.

Abstract. Starting from a permutation of $\{0, \ldots, n-1\}$ we compute in parallel with a workload of $O(n \log n)$ a compact data structure of size $O(n \log n)$. This data structure allows to obtain the associated permutation graph and the transitive closure and reduction of the associated order of dimension 2 efficiently. The parallel algorithms obtained have a workload of $O(m + n \log n)$ where m is the number of edges of the permutation graph. They run in time $O(\log^2 n)$ on a CREW PRAM.

1 Introduction

Permutation graphs are combinatorial objects that found a lot of attention in recent years. This interest led to many results under a structural point of view as well as algorithmically, see [1, 2, 6, 7, 8].

By definition permutation graphs have a compact encoding of size n, where n is the number of vertices. In sequential computing model, it is possible to pass from the graph to the permutation and vice versa with a workload of $O(n^2)$. For parallel computing this has been open up to now. In this paper we show how to pass from the permutation to the graph on a CREW PRAM with a workload of $O(m + n \log n)$ where m is the number of inversions of the permutation. This also leads to a new representation of permutation graphs in size $O(n \log n)$.

Many graph algorithms require a classical representation of the graph. In order to run these algorithms on a permutation graph, it is necessary to compute the graph from the permutation.

The whole article is written in the directed context. We call permutation digraphs the directed version of permutation graphs. They are also called the orders of dimension 2. The results we give in the last section are specific to the directed context. All the others can obviously be obtained for permutation graphs.

The paper is organized as follows.

* The authors are supported by PROCOPE.
** Email: gustedt@math.tu-berlin.de
*** Email: morvan@litp.ibp.fr
† Email: lavie@litp.ibp.fr

After giving the necessary definitions and notations in Sect. 2, we start by computing the number of arcs of the permutation graph. The workload for this computation is $O(n \log n)$ and thus a factor of $\log \log n$ away from the lower bound [4, 5].

In the following section we describe a compact data structure derived from that algorithm that represents the successors sets of all elements in size $O(n \log n)$. In contrast to the one given directly by the permutation this one has the advantage of supporting all classical algorithms on permutation graphs where they are supposed to be represented by adjacency lists.

In Sect. 5 we show how to compute the adjacency lists from our compact representation with a workload of $O(m + n \log n)$.

In the last section we consider the permutation graph as a two-dimensional order and we show how to compute the transitive reduction of the given order from our algorithm.

The problem of computing efficiently representations of an order of dimension d was introduced in these terms by Spinrad in [9]. These problems can be solved with geometric range queries in a d-dimensional space. But this method uses a strong computational model with reals. Moreover it hides the new data structure for representing digraphs that we introduce in Sect. 4. We give here the benefits of a parallel approach to the sequentially well know case of $d = 2$.

2 Definitions and Notations

We always consider simple finite digraphs with no loops.

A *digraph* $G = (V, \mathcal{A})$ is given by a set V of *vertices*, and a set $\mathcal{A}$ of couples of vertices called *arcs*.

If $(u, v) \in \mathcal{A}$, we say that v is a *successor* of u and with respect to undirected graphs terminology that u and v are *adjacent*. Usually, a digraph is represented as *adjacency lists*. For each vertex v, the list of its successors is given. We call this a *classical representation*.

The *degree* $Deg^+(v)$ of a vertex v is the number of its successors.

A permutation π is a bijection from $\{0, \ldots, n-1\}$ into itself, or equivalently a simple word with all the letters $0, \ldots, n-1$. Let $\pi(i)$ denote the image of i, or equivalently the ith letter of π. We write π^{-1} for the inverse of the bijection π, and $\widetilde{\pi}$ for the reverse of the word π.

We will always consider digraphs with set of vertices $\{0, \ldots, n-1\}$.

A digraph $G = (V, \mathcal{A})$ is a *permutation digraph* if $\mathcal{A}$ is given by a permutation π of $\{0, \ldots, n-1\}$ such that $(i, j) \in \mathcal{A}$ iff $i < j$ and $\pi(i) < \pi(j)$.

A permutation digraph $G = (V, \mathcal{A})$ is acyclic and transitively closed. It can be seen as an order of dimension two. Its *transitive reduction* is the minimal subgraph whose transitive closure is G. It is equivalently given by the covering relation on V. We say that u is *covered* by v iff $(u, v) \in \mathcal{A}$ and there is no k such that $(u, k) \in \mathcal{A}$ and $(k, v) \in \mathcal{A}$.

3 Computing the Number of Arcs

The number of arcs of the permutation digraph is the number of inversions of $\widetilde{\pi^{-1}}$ since an arc (i,j) with $i<j$ exists iff i appears before j in π^{-1}.

We are going to compute this number by sorting $\widetilde{\pi^{-1}}$ in a "quicksort way". In each dividing phase of quicksort we update a counter c such that the sum of c and the number of inversions of the permutation being sorted remains invariant.

W.l.o.g. we assume that $n=2^{q+1}$. Since we are sorting exactly all the numbers $0,\ldots,n-1$, we can always find a good pivot element that equally divides the sequences in question. Thus the algorithm requires only a logarithmic number of phases $\varphi=0,\ldots,q-1$.

3.1 Partitionning the Permutation

Now we describe how to partition a part of size $2^{q-\varphi}$ of the permutation at phase φ into a sequence of two blocks, detecting some of its inversions without creating any new one.

Like in quicksort, the elements greater (resp. smaller) than the pivot element are put in the right (resp. left) block. Thereby the order of the permutation in each block has to be preserved. Doing this we cancel some inversions. For each vertex v going to the left, we have to count how many vertices previously appearing to its left go to the right and add this number to the counter c, see Fig. 1.

Before the initial phase, we set $W:=\widetilde{\pi^{-1}}$ and $c:=0$. In phase φ, we partion the 2^{φ} consecutive blocks of size $2^{q-\varphi}$ composing W. For the sake of clarity, we just write the algorithm for the first block.

Algorithm 1 Divide and Conquer Phase.

Input: A block $W(0),\ldots,W(2^{q-\varphi}-1)$ of a permutation.
Output: A number $\Delta(W(x))$ of newly detected inversions relatively to each vertex $W(x)$ for $x\in\{0,\ldots,2^{q-\varphi}-1\}$.

Step 1. **Comparisons:** Every vertex $W(x)$ is compared to the pivot element $2^{q-\varphi-1}$. $B(x)$ is set to 0 if $W(x)$ is smaller and 1 otherwise.

Step 2. **Sum up:** Compute the prefix sums $\sum_{i=0}^{x} B(i)$.

Step 3. **Inversions:** If $B(x)=0$, we have newly detected $\Delta(W(x)):=\sum_{i=0}^{x} B(i)$ inversions relatively to $W(x)$
Otherwise, we set $\Delta(W(x)):=0$.

Step 4. **Partition:** Divide the block in a quicksort step.

For the other blocks of phase φ we proceed analogously. We just have to compute an offset for each block.

At the end of phase φ, we update the number c of inversions that were detected by adding the sum of all $\Delta(v)$. This can be computed with a prefix sum.

$\pi =$	2 4 7 0 5 6 1 3	
$\pi^{-1} =$	3 6 0 7 1 4 5 2	
$\widetilde{\pi^{-1}} =$	2 5 4 1 7 0 6 3	$c = 0$
$\varphi = 1$	2 1 0 3 | 5 4 7 6	$c = 0 + 0 + 2 + 3 + 4 = 9$
$\varphi = 2$	1 0 | 2 3 | 5 4 | 7 6	$c = 9 + 1 + 1 + 0 + 0 = 11$
$\varphi = 3$	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7	$c = 11 + 1 + 0 + 1 + 1 = 14$
		$m = c = 14$

Fig. 1. A permutation and the different phases of the algorithm.

3.2 Analysis

At the end, W is sorted and has no inversions, so c is the number of inversions of $\widetilde{\pi^{-1}}$, and we have computed the number of arcs $m = c$ of the permutation digraph.

At each of the $\log n$ phases, we compute two prefix sums, thus requiring a time $O(\log n)$ and a workload of $O(n)$. Hence we have:

Theorem 1. *Our algorithm computes the number of arcs of a permutation digraph given by a permutation on n elements in time $O(\log^2 n)$, and with a workload of $O(n \log n)$.*

This is not far from the best known sequential algorithm which needs time $O\left(\frac{n \log n}{\log \log n}\right)$ [4, 5].

4 A Compact Representation of Adjacency Lists

In fact, the previous algorithm implicitly computes a special representation of the digraph that we introduce now. Therefore we keep copies W_φ and Δ_φ of the vectors obtained in each phase φ, see Fig. 2.

Let us consider the previous algorithm in terms of the permutation digraph. The number $\Delta_\varphi(v)$ of newly detected inversions relative to v corresponds to $\Delta_\varphi(v)$ successors of v. If v goes to a right block, none of its successors is detected. Otherwise v is smaller than the pivot element. Thus the vertices going to the right block are greater than v. And those which in addition appeared on its left before phase φ are among its successors. They are exactly the $\Delta_\varphi(v)$ first elements of the right block and form an interval $I_\varphi(v) = [l_\varphi(v), r_\varphi(v)]$ of W_φ for some indices $l_\varphi(v)$ and $r_\varphi(v)$.

Notice that the other successors of v go to the left block together with v, and will be detected later.

We can compute $I_\varphi(v)$ in phase φ as follows. $l_\varphi(x)$ is the offset calculated for block $W_{\varphi-1}$. We have $r_\varphi(v) = l_\varphi(v) + \Delta_\varphi(v)$. Let x be such that $v = W_{\varphi-1}(x)$ and $x = b_1 \ldots b_q$ be its binary representation. We have $l_\varphi(x) = \underbrace{b_1 \ldots b_\varphi 10 \ldots 0}_{q}$.

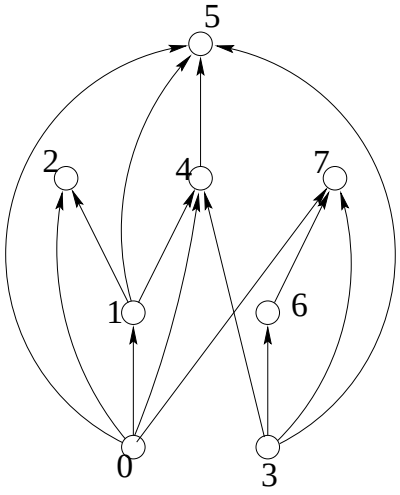
We formalize this concept of representing a digraph by intervals over total orders on V in the following way.

Definition 2. Let $G = (V, \mathcal{A})$ be a digraph, and for some integer k let $W_1, \ldots, W_k$ be arrays representing totally ordered subsets of V. For each vertex $v \in V$, let $I_1(v), \ldots, I_k(v)$ be intervals of $W_1, \ldots, W_k$ respectively. $I_\varphi(v) = [l_\varphi(v), r_\varphi(v)]$ is the set $\{W_\varphi(l_\varphi(v)), W_\varphi(l_\varphi(v)+1), \ldots, W_\varphi(r_\varphi(v))\}$.

We say that $W_1, \ldots, W_k$ and $I_1, \ldots, I_k$ form a *k-compact interval representation* of G if the successor set $Succ(v)$ of each vertex $v \in V$ is given by $Succ(v) = \bigcup_{i=1}^{k} I_i(v)$.

Every graph with n vertices has a trivial n-compact interval representation by setting W_v to the list of successors of v, and by setting $I_v(v) = [1, |W_v|]$ and $I_\varphi(v) = \emptyset$ if $\varphi \neq v$.

We have seen how to compute a k-compact representation of any permutation digraph for $k = \log n$, see Fig. 2. Clearly it also runs in time $O(\log^2 n)$ and with a workload of $O(n \log n)$.



	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
W_0	2	5	4	1	7	0	6	3
W_1	2	1	0	3	$]_2$5	4$]_1$	7$]_0$	6$]_3$
W_2	1	0	$[$2$]_1$	$]_0$3	5	$[$4$]_5$	$]_4$7	6
W_3	0	$[$1$]_0$	2	$]_2$3	4	$[$5$]_4$	6	$[$7$]_6$

	Δ_1	Δ_2	Δ_3	I_1	I_2	I_3
0	3	1	1	$[4, 6]$	$[2, 2]$	$[1, 1]$
1	2	1	0	$[4, 5]$	$[2, 2]$	$\emptyset$
2	0	0	0	$\emptyset$	$\emptyset$	$\emptyset$
3	4	0	0	$[4, 7]$	$\emptyset$	$\emptyset$
4	0	0	1	$\emptyset$	$\emptyset$	$[5, 5]$
5	0	0	0	$\emptyset$	$\emptyset$	$\emptyset$
6	0	0	1	$\emptyset$	$\emptyset$	$[7, 7]$
7	0	0	0	$\emptyset$	$\emptyset$	$\emptyset$

Fig. 2. The permutation digraph of the permutation of Fig. 1 and the intervals of its compact representation.

5 A Parallel Algorithm Which Finds a Classical Representation from a Compact Representation

In order to compute the classical representation of a permutation digraph, we will give in this section an algorithm for finding a classical representation of any digraph from a compact representation. It will lead us to the following results:

Theorem 3. *Let G be a digraph on n vertices and m arcs in a k-compact representation. There is an algorithm that computes a classical adjacency list representation in time $O(\log n)$ and with a workload of $O(m + k.n)$ on a CREW PRAM.*

Combining this with the previous sections, we obtain:

Theorem 4. *It is possible to compute the classical adjacency lists representation of a permutation digraph in time $O(\log^2 n)$ and with a workload of $O(m+n \log n)$ on a CREW PRAM.*

Arc adjacency list representations have size m where m is the number of arcs of the graph. We first have to compute this number which will also be the number of processors our algorithm will require.

Algorithm 2 Outline of the Algorithm.

Input: k arrays $W_1, \ldots, W_k$ and an array of size nk of intervals
$I_\varphi(v) = [l_\varphi(v), r_\varphi(v)]$, for $0 \leq x < n$ and $1 \leq \varphi \leq k$.
Output: An adjacency lists representation.
Step 1. Compute the number m of arcs.
Step 2. Allocate an array A of size m; each element of A will contain an arc of the digraph.
Step 3. Compute the tail of each of these arcs.
Step 4. Compute the head of each of these arcs.

For the following detailed algorithms, recall that for all graphs the set of vertices is $\{0, \ldots, n-1\}$.

Algorithm 3 Computing the Number of Arcs.

Step 1. For each interval $I_\varphi(v)$, set $\Delta_\varphi(v)$ to its length.
Step 2. Let D be an array of size nk.
Step 3. For all $1 \leq \varphi \leq k$ and $0 \leq v \leq n-1$, set $D(kv + \varphi) := \Delta_\varphi(v)$.
Step 4. Compute the prefix sums $S_\varphi(v) := \sum_{a=0}^{kv+\varphi-1} D(a)$
Step 5. Set $m := S_{k+1}(n-1)$.

Observe that the values $S_\varphi(v)$ computed in step 4 are equal to $\sum_{u=0}^{v-1} Deg^+(u) + \sum_{\psi=1}^{\varphi-1} \Delta_\psi(v)$.

Algorithm 4 Computing the Tails of the Arcs.

Output: The sorted array A.tail of the tails of the graph's arcs.
Step 1. Initialize an array T of size m to 0.
Step 2. **Mark** T**:** **for all** $0 \leq v < n$ **do** $T\left(S_1(v)\right) := 1$
Step 3. **tails:** Compute the prefix sums $A.\text{tail}(a) := \sum_{b=0}^{a} T(b)$.

Now the degree of a vertex v can be found as $S_{k+1}(v) - S_1(v)$. A.tail now looks as follows:

$$A.\text{tail} = \underbrace{0 \cdots 0}_{Deg^+(0)}\ \underbrace{1 \cdots 1}_{Deg^+(1)} \cdots\cdots \underbrace{n-1 \cdots n-1}_{Deg^+(n-1)}$$

Algorithm 5 Computing the Heads of the Arcs.

Output: The sorted array A.head of the heads of the graph's arcs.
Step 1. Initialize an array A.phase of size m to 0.
Step 2. **Mark** A.phase**:** **for all** $0 \leq v < n$ **and** $1 \leq \varphi \leq k$ **do**
$A.\text{phase}(S_\varphi(v)) := 1$
Step 3. **Interval:** Compute the prefix sums $A.\text{phase}(a) := \bigoplus_{b=0}^{a} A(b)$
Step 4. **heads:** **for all** $0 \leq a < m$ **do**
$v := A.\text{tail}(a)$
$\varphi := A.\text{phase}(a)$
$A.\text{head}(a) := W_\varphi\left(l_\varphi(v) + a - S_\varphi(v)\right)$.

The operation $\oplus$ is used to compute prefix sums in each adjacency list block. With $A(a) = \big(A.\text{tail}(a), A.\text{phase}(a)\big) = (u, \varphi)$ and $A(b) = (v, \psi)$, it is defined by $A(a) \oplus A(b) = (v, \psi)$ if $u < v$ and $A(a) \oplus A(b) = (u, \varphi + \psi)$ if $u = v$. Thus after step 3 A.phase looks as follows:

$$A.\text{phase} = \underbrace{\underbrace{1 \cdots 1}_{\Delta_1(0)} \cdots \underbrace{k \cdots k}_{\Delta_k(0)}}_{Deg^+(0)}\ \underbrace{\underbrace{1 \cdots 1}_{\Delta_1(1)} \cdots \underbrace{k \cdots k}_{\Delta_k(1)}}_{Deg^+(1)} \cdots\cdots \underbrace{\underbrace{1 \cdots 1}_{\Delta_1(n-1)} \cdots \underbrace{k \cdots k}_{\Delta_k(n-1)}}_{Deg^+(n-1)}$$

The complexity of the algorithm comes from the prefix sum's one. Note that only the last step requires a concurrent read.

6 Computing the Transitive Reduction of a Permutation Digraph

Let us finish by a similar algorithm for computing the transitive reduction. In contrast to the previous one, among the successors of a vertex, we only want to select those corresponding to arcs of the transitive reduction, see Fig. 3.

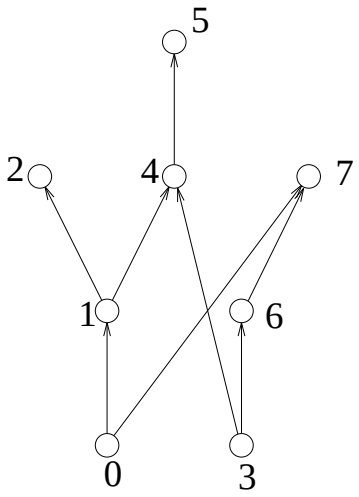


Fig. 3. The transitive reduction of the permutation digraph of Fig. 2.

An arc (i, j) of a permutation digraph is in the transitive reduction if in addition there is no k such that $i < k < j$ and $\pi(i) < \pi(k) < \pi(j)$, or equivalently if there is no k such that $i < k < j$ appearing between i and j in π^{-1}.

Let S_i be the list of the successors of i in the same order as in π^{-1}. The arcs (i, j) of the transitive reduction are those verifying $j \in S_i(p)$ and $j \geq S_i(q), \forall q < p$, or equivalently:

$$j \in S_i \qquad \text{and} \qquad j = \min S_i(1), S_i(2), \ldots, j \ .$$

If the adjacency lists of the transitive closure are given and ordered according to π^{-1}, a prefix sum over all these lists allows us to compute these minimums. Then a test of equality between a successor and this minimum determines whether it corresponds to an arc of the transitive reduction. This requires time $O(\log n)$ and a workload of $O(m)$ where m is the number of arcs of the transitive closure.

Since we can sort the adjacency lists in time $O(\log n)$ with a workload of $O(m \log n)$ [3], we obtain:

Theorem 5. *It is possible to compute the transitive reduction of a permutation digraph in time $O(\log^2 n)$ and a workload of $O((n+m) \log n)$ on a CREW PRAM.*

References

1. K.A. Baker, P.C. Fishburn, and F.S. Roberts. Partial orders of dimension 2. *Networks*, 2:11–28, 1971.

2. C.J. Colbourn. On testing isomorphism of permutation graphs. *Networks*, 11:13–21, 1981.
3. Richard Cole. Parallel merge sort. *SIAM J. Comput.*, 17(4), august 1988.
4. Paul F. Dietz. Optimal algorithms for list indexing and subset rank. In *Algorithms and data structures, Proc. workshop WADS '89, Ottawa/Canada*, number 382 in Lect. Notes Comput. Sci., pages 39–86, 1989.
5. M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *21st ACM STOC*, pages 345–354, 1989.
6. M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1985.
7. A. Pnueli, A. Lempel, and W. Even. Transitive orientation of graphs and identification. *Canad. J. Math.*, 23:160–175, 1971.
8. J. Spinrad and J. Valdes. Recognition and isomorphism of two dimensional partial oreders. In *10th Coll. on Automata, Language and Programming*, number 154 in Lect. Notes Comput. Sci., pages 676–686, Berlin, 1983. Springer.
9. Jeremy Spinrad. Dimension and algorithms. In V. Bouchité and M. Morvan, editors, *Orders, Algorithms, and Applications*, number 831 in Lect. Notes Comput. Sci., pages 33–52. Springer-Verlag, 1994.